

Formulierungshilfen

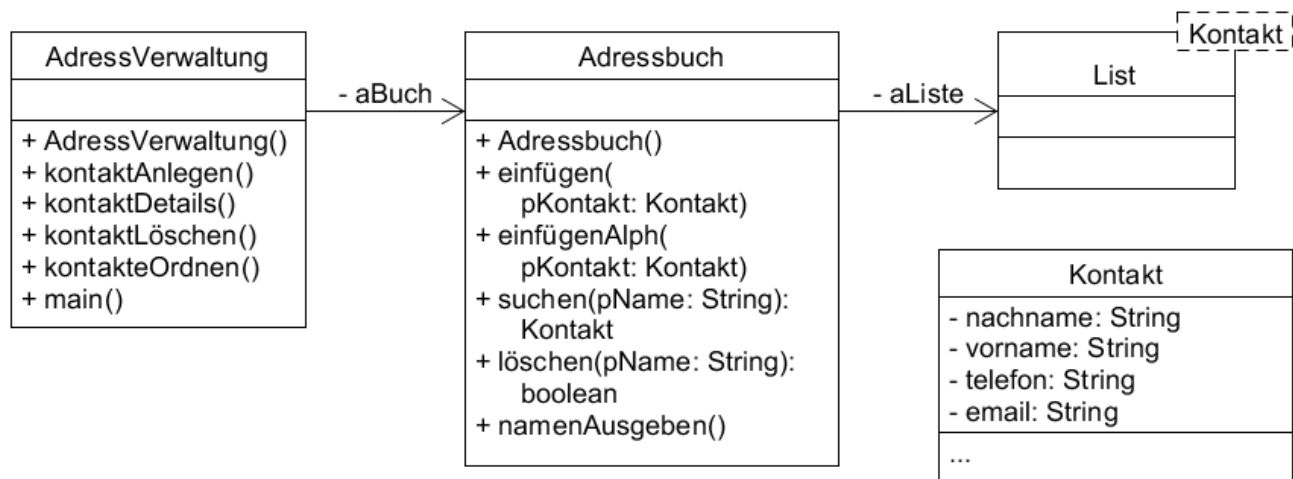
Es ist wesentlich leichter, auf einzelne Elemente einer Liste zuzugreifen, diese zu verschieben usw. als in einer Queue oder einem Stack. Daher kann die Liste für ein breiteres Spektrum von Anwendungen eingesetzt werden. Da die Klasse List auch weitaus mehr Methoden als Queue und Stack zur Verfügung stellt, ist die Formulierung von Algorithmen eine größere Herausforderung.

Um Algorithmen in Umgangssprache zu formulieren, nutze folgende Formulierungshilfen:

<code>if (list.isEmpty())</code>	falls die Liste leer ist
<code>list.toFirst()</code>	wähle das erste Element der Liste aus
<code>while (list.hasAccess())</code>	wiederhole, solange ein Element ausgewählt ist
<code>list.next()</code>	wähle das nächste Element der Liste aus
<code>list.getContent()</code>	das ausgewählte Element
<code>list.setContent(...)</code>	ersetze das ausgewählte Element durch ...
<code>list.insert(...)</code>	füge ... vor dem ausgewählten Element ein
<code>list.append(...)</code>	füge ... am Ende der Liste ein
<code>list.remove()</code>	entferne das ausgewählte Element aus der Liste

Anwendungssituation

Die Aufgaben beziehen sich auf das im vorigen Abschnitt implementierte Adressbuch. Es werden Verbesserungen und zusätzliche Funktionen hinzugefügt.



Aufgabe 1

Der folgende Quellcode zeigt die Implementierung der Methode löschen() der Klasse Adressbuch. Gib den entsprechenden **Algorithmus** in Umgangssprache an.

```
public boolean löschen(String pName)
{
    Kontakt kAkt;
    aListe.toFirst();
    while (aListe.hasAccess() == true)
    {
        kAkt = aListe.getContent();
        if (pName.equals(kAkt.getNachname()))
        {
            aListe.remove();
            return true;
        }
        aListe.next();
    }
    return false;
}
```

Aufgabe 2

Für die Anzeige aller Namen und für die Suche innerhalb der Liste ist es sinnvoller, wenn die Kontakte in alphabetischer Reihenfolge in der Liste gespeichert werden. Geordnet werden soll die Liste nach den Nachnamen der Kontakte.

Gib für die jeweilige Methode erst einen **Algorithmus** in Umgangssprache an.

Implementiere bzw. ändere dann die Methoden der Klasse Adressbuch.

a) void einfügenAlph(Kontakt pKontakt)

Der neue Kontakt pKontakt soll nicht einfach am Ende der Liste, sondern an der alphabetisch richtigen Stelle eingefügt werden (unter der Annahme, dass die Liste immer sortiert ist).

Hilfestellung zur Implementierung: Ob ein String alphabetisch „größer“ oder „kleiner“ als ein anderer String ist, kann man mithilfe der compareTo-Methode herausfinden:

str1.compareTo(str2) < 0 → str1 steht im Alphabet VOR str2

str1.compareTo(str2) = 0 → beide stehen an der gleichen Stelle (d.h. sind gleich)

str1.compareTo(str2) > 0 → str1 steht im Alphabet HINTER str2

b) Kontakt suchen(String pName)

Die Methode wird etwas optimiert: In der sortierten Liste wird die Suche abgebrochen, sobald sicher ist, dass der gesuchte Name nicht in der Liste enthalten sein kann.

c) Das Attribut aListe der Klasse Adressbuch ist privat. Begründe, warum die Klasse Adressbuch keinen direkten Zugriff auf die Datenstruktur Liste für andere Klassen (auch nicht für die Klasse AdressVerwaltung) erlauben sollte.

Aufgabe 3 (angelehnt an eine Abituraufgabe von 2015)

Der folgende Algorithmus führt eine Funktion für die Liste aListe der Klasse Adressbuch aus:

Setze eine Variable a auf 0.

Wähle den ersten Kontakt von aListe aus.

Wiederhole, solange ein Kontakt von aListe ausgewählt ist: (← äußere Schleife)

Setze eine Referenz min auf den ausgewählten Kontakt.

Wähle den nächsten Kontakt von aListe aus.

Wiederhole, solange ein Kontakt von aListe ausgewählt ist:

Falls der Name des ausgewählten Kontakts alphabetisch vor dem Namen von min ist,

Setze min auf den ausgewählten Kontakt.

Wähle den nächsten Kontakt von aListe aus.

Wähle den ersten Kontakt von aListe aus.

Wiederhole, solange der ausgewählte Kontakt nicht gleich min ist:

Wähle den nächsten Kontakt aus.

Entferne den ausgewählten Kontakt.

Wähle den ersten Kontakt von aListe aus.

Wiederhole a mal:

Wähle den nächsten Kontakt von aListe aus.

Falls ein Kontakt von aListe ausgewählt ist:

Füge min vor dem ausgewählten Kontakt ein.

Sonst:

Füge min am Ende von aListe ein.

Erhöhe a um 1.

- a) Analysiere den Algorithmus, indem du ihn auf die folgende Liste von Kontakten anwendest (hier nur die Nachnamen gegeben).
Gib dabei die vollständige Liste am Ende jeder Wiederholung der äußeren Schleife an.

Picasso	Dalí	Kandinsky	Cézanne	Monet
---------	------	-----------	---------	-------

- b) Gib an, welche Funktion der Algorithmus hat.
Welcher (dir schon bekannte) Algorithmus wird hier angewendet?
- c) Erläutere, welche Funktion die Variable a in diesem Algorithmus hat.
- d) Implementiere die entsprechende Methode in der Klasse Adressbuch.
- e) Erweitere die Klasse AdressVerwaltung, so dass du die Methode testen kannst.
Hinweis: Verwende beim Testen die einfache Methode `einfügen()`, die neue Kontakte am Ende der Liste einfügt.